Regular Paper

Systems Requirements Engineering—State of the Methodology

Yvonne Bijan,[1,*] Junfang Yu,[2] Jerrell Stracener,[2] and Timothy Woods[2]

[1]*Southern Methodist University, 1608 Briar Run, Benbrook, TX 76126*
[2]*Southern Methodist University, Dallas, TX 75205*

Received 27 July 2011; Revised 19 May 2012; Accepted 19 May 2012, after one or more revisions
Published online 23 October 2012 in Wiley Online Library (wileyonlinelibrary.com).
DOI 10.1002/sys.21227

ABSTRACT

There are many references about the characteristics of a good requirement. But what is the process for developing clear, unambiguous requirements, and how do we know when we have defined the requirements successfully? This paper investigates the current state of the methodology for developing complex system requirements. Significant work has been accomplished over the last several years to describe requirements development and systems engineering. This paper identifies and investigates requirements development methodologies and techniques. © 2012 Wiley Periodicals, Inc. Syst Eng 16: 267–276, 2013

Key words: systems engineering; requirements; methodology

1. INTRODUCTION

The focus of this paper is a survey of the methods used for requirements development. Requirements development is a transformation [Balmelli et al., 2008] of the concerns of the customer into a specification that can serve as a basis for developing a system. The question is how should this transformation be performed to yield the best results? Many people believe requirements must inevitably change, and some seem to believe they must change frequently. Reasons for changing requirements include incompletely captured requirements [Errey, 2007], over specification, under specification [Bittner, 2008], changing threats during systems development, engineers rushing through the requirements engineering process [Laplante, 2009], technology changes, personnel changes, and scope creep.

*Author to whom all correspondence should be addressed (e-mail: Yvonne_Bijan@hotmail.com; yuj@lyle.smu.edu; jerrells@lyle.smu.edu; twoods@smu.edu).

Systems Engineering Vol. 16, No. 3, 2013

The goal of requirements development methodologies is to develop a set of system level requirements. Developing system level requirements includes eliciting the customer needs, prioritizing the needs, and translating those needs into a specification that includes a complete unambiguous set of requirements. According to Hull, Jackson, and Dick [2011], it is essential to establish the need for the system before developing it. Determining the customer needs involves facilitating exercises with the customer to determine what is really needed. There are many ways to facilitate need definition with the customer. Among these are Quality Function Deployment (QFD), The Five Whys, and Use Case Modeling. Once the needs are understood, they need to be prioritized by the customer in order for the contractor to understand the ranking by importance. Engineers can use prioritized needs to perform trades with requirements and design solutions. There are ample number of ways to prioritize requirements including Analytical Hierarchy Process (AHP), Borda's method, and Wiegers' Prioritization Matrix. While elicitation and prioritization techniques are defined, the translation of needs to requirements is more difficult to define. There are many resources that discuss defining requirements. These generally include a list of good characteristics a requirement

must have. But what is the process for developing clear, unambiguous requirements and measuring when a requirement is defined successfully? For some time now, people have been provided anecdotal examples of good and bad requirements [Wiegers, 1999], yet there seems to be little definition for the quantification of a good requirement. Even though people think they know a good requirement when they see one, many engineers cannot articulate how to write or how to measure a good requirement. Kasser, Weiss, and Hari [2006] propose several metrics for evaluating requirements. Kasser et al.'s metrics include usage of poor words, wrong words, multiple requirements in one statement, and unverifiable requirements which feed into the figure of merit metric. Kasser et al. go so far as to include a list of words that should not be used in requirements. Indeed, no matter how complex the situation, good systems engineering [Gilb, 2005] involves putting value measurements on the important parameters of desired goals and performance of pertinent data and of the specifications of the people and equipment and other components of the system.

A survey of IT projects performed by The Standish Group [1995] found that the second and third leading causes for challenged projects involved requirements. The leading cause involved lack of user inputs. For failed projects, the leading cause was requirement issues. The NDIA has been investigating Systems Engineering issues and publishing reports on the top five issues periodically. In 2003 [NDIA, 2003], the NDIA task group reported requirements definition, development, and management application as issue number four. Again in 2006 [NDIA, 2006], the group reported issue three as poorly managed requirements including ineffective translation from capabilities to requirements. In 2010 [NDIA, 2010], there had been some improvement in requirements development; however, the task group reported "variability in approaches to requirements definition, validation, and consolidation continue" (p. 2). Requirement issues were not called out as a standalone issue in 2010 anymore, but were called out in connection with the third as the group noted "programs do not always start with good requirements/capabilities" (p. 6) and "programs do not always capture evolving requirements adequately" (p. 6). Additionally, 80% of major US DoD acquisition programs fail to meet cost and schedule goals, and 47 programs have been through Nunn-McCurdy [Sledge, 2011]. Under the Nunn-McCurdy Provision [10 U.S.C. § 2433a(b)], programs are required to notify Congress if cost increases by more than 15%. Programs exceeding 25% cost increases run the risk of being cancelled.

Indubitably, many problems on a project are a result of poor requirements. See Figure 1 for a comprehensive breakdown of where errors start in a program and their categorization. The first number in the Vee is the percentage of problems that occurred in a given phase. The second number is the percentage of problems fixed in that phase and the last number is the relative cost of fixing the problems in a given phase. As can be seen in the figure, the largest number of problems are initiated in the left side of the Vee and the prevalent category for the faults is requirements.

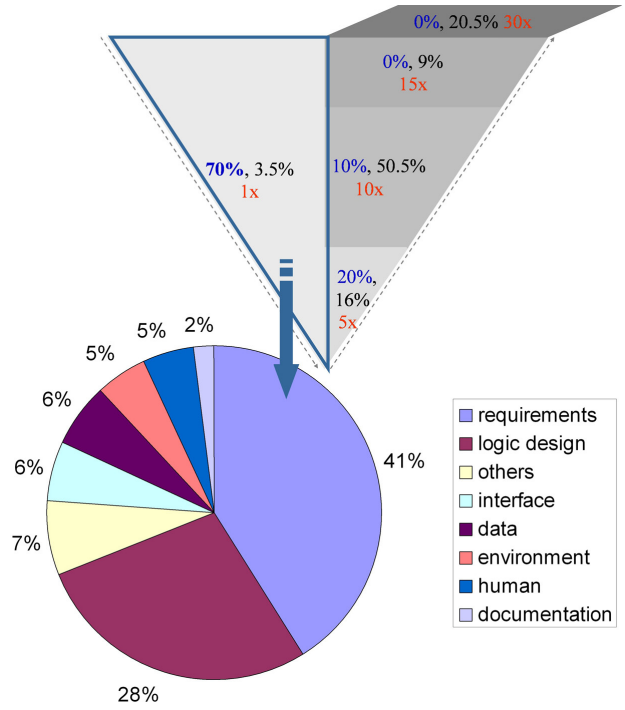

Figure 1. Share of faults per category [Hood et al., 2008; Sheldon et al., 1992]. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

1.1. The Basic Definitions

There are many definitions for requirements. In this article, we use a standard definition for requirements from IEEE Std 610.12-1990 Standard Glossary of Software Engineering Terminology [IEEE, 1990]:

- A requirement is a condition or capability needed by a user to solve a problem or achieve an objective.
- A requirement is a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formality.

INCOSE [2011b] states the purpose of the stakeholder requirements definition process is to define the requirements for a system that can provide the services needed by users and other stakeholders in a defined environment.

Pohl [2010] defines requirements engineering as a cooperative, iterative, and incremental process which aims at ensuring:

- All relevant requirements are explicitly known and understood at the required level of detail.
- A sufficient agreement about the system requirements is achieved among the stakeholders involved.
- All requirements are documented and specified in compliance with the relevant documentation/specification formats and rules.

1.2. Characteristics of Good Requirements

There are many definitions of what constitutes a good requirement. Ivy Hooks [1993: 2] wrote: A good requirement states something that is necessary, verifiable, and attainable. The list below, based on Hood et al. [2008], is a concise list of characteristics which is similar to many other sources:

- Complete
- Unambiguous
- Understandable
- Identifiable
- Free of duplication
- Traceable to source
- Free of contradiction
- Feasible
- Verifiable
- Atomic
- Correctly derived.

Other lists with similar characteristics can be found in *Software Requirement: Objects, Functions, and States* [Davis, 1993], *Requirements Engineering* [Hull, Jackson, and Dick, 2011], *Requirements Engineering: From System Goals to UML Models to Software Specifications* [van Lamsweerde, 2009], "Writing Good Requirements" [Hooks, 1993], IEEE 830 [IEEE, 1998], *Systems Engineering Fundamentals Guide* [DAU, 2001], *Writing Quality Requirements* [Wiegers, 1999], and "A Hybrid Requirements Capture Process" [Daniels, Bahill, and Botta, 2005].

2. REQUIREMENTS DEVELOPMENT METHODS

There are a number of methods for developing requirements. The following methods are broken down into facilitation, prioritization, and transformation groupings, with a number of standalone methodologies also presented.

2.1. Facilitation

The first step of any successful system design process [Childers and Long, 1994] is understanding the needs and desires of the acquisition agency and understanding the context in which the system will operate. During this step, engineers, analysts, and the customer need to ensure that requirements are implementation-free. People can capture implementation details as constraints if needed. Methods for facilitation and prioritizing customer needs are well understood even if not consistently applied.

2.1.1. Quality Function Deployment (QFD) Affinity Diagrams

QFD provides a structure to the front end process of the requirements definition stage. Thompson and Fallah [1989] define it as a systematic approach for identifying and ranking customer needs and translating those needs into product/service specifications.

The first step is to perform brainstorming activities to define some stakeholder needs. The second step is to develop affinity diagrams which provide systems engineers with a visual method for determining if the groupings are complete and free of redundant needs. See Figure 2 for an example affinity diagram. Affinity diagrams can be used to group anything and helps systems engineers categorize needs to fit within capabilities.

2.1.2. The Five Whys

Sometimes, customers tell the engineers they want a particular implementation. The implementation may address the symptoms of a problem they are facing; however, it is important for engineers to understand what the problem is. This can be dealt with by asking the customer a series of "whys." The point is to get to the real need behind the request for the implementation. Once the real need is understood, the engineers can write proper requirements and determine the best design.

Below is an example of The Five Whys from The Lean Enterprise Institute Forum [LEIF, 2011], which is included here to illustrate using the Five Whys to get to the root of the problem:

Supervisor: The production line stopped moving.
Foreman: Why did it stop?
Supervisor: The motor that drives the line quit working.
Foreman: Why did it quit working?

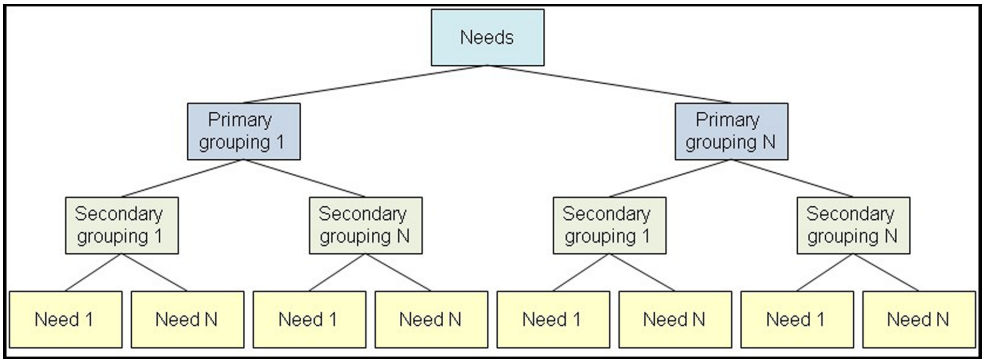

Figure 2. Generic affinity diagram. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Supervisor: A bearing in the motor froze causing a circuit breaker to pop open.
Foreman: Why did the bearing freeze?
Supervisor: Because it didn't have any lubricant and, as a result, heated up until it froze.
Foreman: Why didn't it have any lubricant?
Supervisor: Because maintenance hasn't been lubricating it.
Foreman: Why not?
Supervisor: Because to lubricate the bearings the line has to be shut down for safety reasons and with production running three shifts, the line never shuts down.

2.1.3. Use Cases

Software engineers have been using use cases successfully for years. Some engineers have proposed using use cases for systems engineering to assist with requirements development. Daniels, Bahill, and Botta [2005] point out that standalone requirements make it difficult for people to understand the context and dependencies among the requirements, especially for large systems. Daniels et al. suggest using use cases to define scenarios that provide value to at least one actor.

Hoffmann [2010] recommends tracing requirements to use cases within a model to determine if there is a complete set. Requirements that are not mapped to a use case indicate functionality that is missing from the system while use cases that are not mapped to a requirement indicate functionality that is not necessary or potentially missing requirements. See Figure 3 for an example use case diagram.

2.2. Prioritizing Needs

It is important to understand which requirements are the most important to the customer, i.e., the ranking of requirements. Requirements can inherit their prioritization from the priority of the need they were derived from Kasser [2004]. The US DoD has been discussing contractors developing systems faster by delivering the 80% solution now instead of the 100% solution later. Without having the requirements prioritized, it is impossible to determine which 80% to concentrate on. It is also useful to know the prioritization for risk analysis. The consequences of failing to meet a higher priority requirement are worse than failing to meet lower priority requirements. Many times, there are conflicts among the requirements, and only a subset is feasible for development. When the contractor understands what is most important to the customer, various alternative requirement subsets can be analyzed to determine which one provides the best value to the customer. Also, with today's focus on affordability, it may be necessary to scale back on some requirements due to cost. Prioritizing can assist with defining alternatives that stay within cost constraints while giving the customer the most "bang for the buck." Prioritization is also important for determining which requirements need to be collaborated and monitored more often and which requirements are related to the overall system performance.

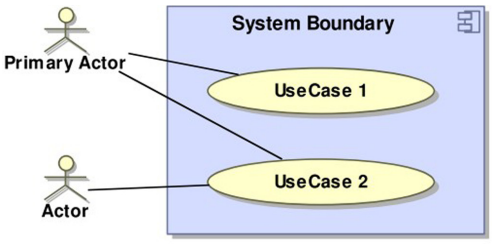

Figure 3. Use case diagram. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Customer needs can be prioritized by simply listing them in the order of most importance. However, this does not allow the contractor to understand the importance of one need over another. Also, requirements may not be independent. The Analytical Hierarchy Process (AHP) is a much more rigorous prioritization methodology [Wagner, 1999]. Needs can also be prioritized using the Wiegers' Prioritization Matrix [Pohl, 2010] or Borda's method [INCOSE, 2011a].

2.2.1. Analytical Hierarchy Process (AHP)

Systems engineers assist the stakeholders with prioritizing their needs using the AHP which involves pairwise comparisons of all of the needs, taking two at a time. The process [Wagner, 1999] involves a systematic procedure for decomposing elements of a decision problem into smaller constituent parts so as to allow simple pairwise comparison judgments to be applied to the elements which results in a prioritization of each element at each level of the decomposed hierarchy. This process was designed to work with a large number of criteria, but does not work well in practice due to its being time-consuming. AHP uses a 1–9 scale with 1 meaning two needs are equally important and 9 meaning one need is extremely more important than another.

The customer should prioritize each of the groupings from the hierarchy developed in the affinity diagram, covered previously, in addition to the individual needs to determine global priorities for each of the needs. The global priorities are the result of multiplying the weights of the needs times the weight of the secondary grouping the need is a part of, times the primary grouping the secondary grouping is a part of.

2.2.2. Wiegers' Prioritization Matrix

Wiegers' method [Pohl, 2010] bases priority on four criteria—benefit, penalty, cost, and risk. Each of the criteria can be weighted differently as needed. Then each requirement is evaluated on the relative benefit of completing the requirement in terms of customer satisfaction, the penalty of not completing the requirement, relative cost of implementing the requirement when compared to other requirements, and relative risk associated with each requirement.

2.2.3. Borda's Method

In the Borda method [INCOSE, 2011a], each decision maker ranks the needs in order of preference. Each decision maker is assigned a rank to represent the importance of the decision maker. The needs are multiplied by the decision maker's rank, and then all of the needs are summed up based on the order of preference from each decision maker and the rank of the decision maker. This method takes into account that some of the people performing the prioritizations outrank others, but it does not account for how much more one need is preferred

over another. It is also impossible to determine where to draw the line for critical versus noncritical needs.

2.2.4. Comparison of Prioritization Methods

Simply listing the needs in order of importance does not capture the relative importance. Engineers will not know if needs one and two are equally important, if need two is slightly less important or significantly less. Borda's method [INCOSE, 2011a] does not capture the relative importance, but it does capture the ranking of the decision makers. AHP yields relative importance, but requires that the needs be grouped into hierarchal categories that are small. It is also very time-consuming to perform a pair-wise comparison on a large number of items. Wiegers' method [Pohl, 2010] is multidimensional and is useful when engineers can gather all of the data needed. If other elements are important for specific programs in addition to value, cost, and risk, such as long lead times or level of effort, the Wiegers' Matrix could be adapted to include them.

2.3. Transforming Gathered Data into Requirements

There are a number of methods that have been employed by software and systems engineers to convert data gathered by facilitation techniques into requirements.

2.3.1. Architecture Tradeoff Analysis

Architecture Tradeoff Analysis [SEI, 2012] is a method for evaluating architectures against quality attribute goals. Architecture Tradeoff Analysis can be used to make requirements more clear. This method involves defining utility trees (needed attributes for the system), defining the architecture, and then analyzing the architecture to ensure it addresses the high priority goals in the utility tree. The analysis allows engineers to explore the relationships among requirements. It can also be used to check for viable solutions. By performing an architectural analysis, engineers will gain insight into the problem and be able to ask the customers questions that they might not have asked.

An example from Thomas [1999: 2] illustrates the usefulness of this method:

> A requirement might call for end-to-end latency of three seconds. It might turn out that it's really hard to achieve that in a particular architecture. We might ask, "Do you really need three seconds?" The stakeholders might say, "No, we just picked that number, it could be five." Or, they might say, "Yes, it's three." Then we just have to change everything to meet three.

2.3.2. Concept Requirement List

Grady [2006] describes a structure for defining requirements. The structure includes defining what needs to be controlled, the value and units of the controlled item, and the relationship between the two, such as equals or less than. Grady recommends using a Concept Requirements List (Fig. 4) so engineers can focus on the content of the requirement first and the sentence structure and grammar later using a conversion process.

DOCUMENT NUMBER
DATE

ITEM REQUIREMENTS

1.	Attribute	Relation	Value	Units
2.	Weight	≤	132	Pounds
3.	Throughput	≥	100	K Bits/Sec

Figure 4. Concept requirements list example [Grady, 2006].

2.3.3. Rapid Prototyping

Colombi and Cobb [2009] describe using informal interviews with users and stakeholders to derive the system requirements in their rapid prototyping methodology. Rapid prototyping follows a modified Vee process. Engineers state the problem, identify critical operating issues, define measures of effectiveness, and then define measures of performance. But the process for getting to an actual requirement is not rigorous. While rapid prototyping does not discuss the requirements definition process, it is a useful method for accomplishing validation of the requirements [Stone et al., 1992]. Prototypes can allow users to observe the behavior of the system and allows for the removal of misunderstandings between users and developers. The prototypes also make it possible for developers to discover defects in the requirements early on in the development process.

2.3.4. Summary and Comparison of Transforming Methods

Architecture tradeoff analysis is appropriate for discovering requirements that are not feasible. It also helps systems engineers discover requirements that relate to the structure of the architecture if they attempt to trace the architecture elements back to the requirements and find gaps. The architecture captures relationships among requirements, but only rudimentary relationships such as "derived from," "parent," "child," and "traces to." It does not help make requirements less ambiguous. The Concept Requirement List allows engineers to focus on what is important in a requirement rather than the grammar. Rapid Prototyping uses interviews to derive requirements which necessitate that people ask the right questions. It does not really cover how to transform the answers from the interviews into requirements.

While some relationships among requirements are captured in these methods, the specific nature of the interdependencies is not, and requirements remain ambiguous and incomplete.

2.4. Model Based Systems Engineering (MBSE)

Balmelli, Nolan, Brown, Bohn, and Wahli [2008] believe that poor development outcomes occur because traditional systems development methods are limited in their capability to address the challenges of operational integration, effective information sharing, and the increasing volatility of requirements and that Model Driven Systems Engineering can address these issues. Karban. Andolfato, and Zamparelli [2009] also recommend using MBSE, stating that, by applying a model based approach, requirements and system interfaces

become much more than simple text as they can be automatically validated and reused.

SysML is commonly used when performing MBSE as it addresses requirements, structure, behavior, and parametrics, which can be used to support engineering analyses. Turk [2005] explains how SysML can be used to support requirements analysis, verification, functional analysis, allocation, and trade studies. Model analysis shows requirements can be met and helps to discover missing requirements, but it does not address documenting requirements as other than textual statements or how to know when there is a complete set. Fisher [1998] states that requirements may also come in the form of relationships, attributes, and diagrams, but US DoD contractors are required to develop specifications with shall statements. Despite this, the models can still be used to determine what these relationships are and to generate the shall statements automatically. A deficiency with SysML requirements is that they are used to represent textual requirements for a system. The authors believe that SysML is capable of doing much more with requirements.

2.5. Planguage

Gilb [2010] states that the problem with systems engineering is that we think like programmers, and not as engineers and managers. Gilb also states that requirements need to include contextual information such as benchmarks, owner, version, stakeholders, gist, ambition, impacts, and supports. Gilb developed a template for defining a function specification called the Planguage Template. The Planguage Template allows engineers to capture information that is relevant to the requirements, helps to clarify the requirement, and justifies why it was included in the specification. It captures relationships among requirements, specifications, and functions. However, these relationships are within a list of each requirement spread out across a number of specifications, making it difficult to perform impact analyses when something needs to change.

2.6. Object-Process Methodology

Object-Process Methodology (OPM) [Dori, 2002] is a holistic method, backed by a graphical and textual language, for modeling standards. While Dori covers using OPM for defining standards in this reference, a similar process can be used for defining requirements. Modeling the behavior of the system ensures the requirements are consistent and less ambiguous. Unlike other techniques, OPM automatically keeps the diagrams in sync with plain English statements. The OPM model integrates structural, functional, and behavioral aspects of a system in a single diagram type [Grobshtein et al., 2007] unlike SysML which spreads the model out into a number of views. OPM [Blekhman, Dori, and Martin, 2011] starts out with a high level system design and then unfolds the abstract diagram into more detailed diagrams.

2.7. Agile

According to the Agile manifesto [Beck et al., 2012], Agile is more focused on customer collaboration than contract negotiation and focuses on responding to changes over following a plan. Instead of documenting all of the requirements up front, there is less effort spent on documentation and more on collaboration with the stakeholders. Agile methods [AlAli and Issa, 2011] include use cases, stories, architecture, and extreme programming which emphasize code over design. Engineers jump right into developing code instead of developing a design. This gives the customers products to evaluate sooner instead of reviewing abstract designs. Agile is ideal [Khurana and Sohal, 2011] in situations with unstable and volatile requirements. Projects are developed using short, numerous iterations instead of one large block. These iterations allow programs to adapt to changing customer needs.

2.8. Lean Systems Engineering (LSE)

Lean [Oppenheim, Murman, and Secor, 2010] focuses on streamlining the flow between the processes and having quality designed in by utilizing a value stream approach and eliminating waste. There are Lean System Enablers added to the systems engineering processes defined in the INCOSE *Systems Engineering Handbook* [INCOSE, 2011b] to improve those processes. The fundamental feature of LSE [Oppenheim, Murman, and Secor, 2011] is to perform all planning well enough to execute tasks right the first time. Value is similar to requirements in that a value definition has the same characteristics as a requirement, such as clear, complete, unambiguous, and necessary to meet customer need. Programs [Oppenheim, Murman, and Secor, 2010] should develop a comprehensive, unambiguous, and detailed understanding of value to the customer (customer needs, requirements, and interpretations of value). LSE is a paradigm rather than a process to write proper requirements. LSE says that value includes unspoken requirements and that these must be captured in order to ensure there is understanding between the stakeholders and developer. LSE [Oppenheim, 2011] uses requirements to define value, but tries to focus on the quality of the effort of requirements development to minimize various pitfalls in traditional requirements development. This paradigm includes definitions for basic principles, but lacks specific guidance for the engineer to follow in performing the enablers.

2.9. Design for Six Sigma

According to Yang and El-Haik [2009], Design for Six Sigma is a process of transforming customers' wants into useful design solutions. The techniques are customer-focused and about designing the solution right the first time. Design for Six Sigma includes a number of processes and methods for identifying requirements, characterizing the design, optimizing the design, and verifying the design. Identifying requirements includes Quality Function Deployment (QFD) techniques and Kano diagrams. Characterizing the design includes translating customer requirements into product requirements. This process spans the facilitation, prioritization, and transformation phases, as well as other Systems Engineering process groups. Design for Six Sigma is an inclusive methodology incorporating many methods including statistical analysis to predict and track how well the design and solution complies with the requirements.

3. ANALYSIS

There are a number of techniques for requirements engineering. Many of the techniques are not independent, and some work better in conjunction with others than when used alone. See Table I for a summary of the methods. Use cases are good for capturing what the stakeholders expect the system to do for them and how they will interact with the system. Even though they do not capture all types of requirements, it is worth taking the time to develop the use cases and make sure all of the customer needs are covered by requirements. The affinity diagram is another method for interacting with the stakeholders to try to capture what they need the system to do. Categorization helps engineers and customer see if there is anything missing from the list. When something does not fit into a category and a new category is created, people may realize there are other items that also need to go in the new category. People may also realize an entire category is missing when they try to lay out everything together. The Five Whys can be used at any point to get clarification from stakeholders about what they really need. AHP is useful when there are not too many items that have to be prioritized. It follows naturally from the affinity diagrams since the grouping from that method can be used to define the hierarchies. AHP is performed with stakeholders in order to understand what is most important to them. However, there may be times when a

Table I. Summary of Methods

Category	Method	Pros	Cons
Facilitation	Use Cases [Daniels, Bahill, and Botta, 2005]	-Captures stakeholders expectations for the system -Understandable	-Does not capture all types of requirements -Ambiguous
	Affinity Diagram [Thompson and Fallah, 1989]	-Categorization of needs help engineers and stakeholder discover missing items.	-Does not help engineers find all requirements
	Five Whys [LEIF, 2011]	-Facilitates getting to the root need of customer statements	
Prioritization	AHP [Wagner, 1999]	-Prioritizes needs and assists in converting subjective rankings into objective ones	-Time consuming, one dimensional, and only supports a limited number of items
	Wiegers' method [Pohl, 2010]	-Multidimensional and adaptable	-Lowers priority on high risk and cost needs which may be critical to the customer
	Borda's method [INCOSE, 2011a]	-Accounts for rank of decision makers	-Does not determine how much more one need is preferred over another
Transformation	Architecture Tradeoff Analysis [Thomas, 1999]	-Shows relationships among requirements	
	Concept Requirements List [Grady, 2006]	-Provides a template -Focus is on content instead of grammar	
	Rapid Prototyping [Colombi and Cobb, 2009]	-Fast -Good communication with users	-Lack of rigor -Misses requirements
Other	MBSE [Turk, 2005; Karban, Andolfato, and Zamparelli, 2009]	-Relationships among requirements and system	-Textual requirement statements
	Planguage [Gilb, 2010]	-Contextual information regarding requirements	
	OPM [Dori, 2002]	-Single diagram type -Consistent text and graphics -Reduces ambiguity	
	Agile [Beck et al., 2012; Khurana and Sohal, 2011]	-Adaptable to changing customer needs	-Lack of documentation
	Lean SE [Oppenheim, 2011]	-Optimizes value to customer	-High level enablers; not specific implementation methods
	Design for 6 Sigma [Yang and El-Haik, 2009]	-Includes methods for SE processes -Statistical analyzes	

company wants to prioritize needs based on more than one attribute. Wiegers' prioritization [Pohl, 2010] can be used for those cases with attributes such as value to the customer, risk, and cost.

Architecture Tradeoff Analysis [Thomas, 1999] allows engineers to see how requirements relate to each other. Many requirements are not independent, and design solutions that improve on the subject of one requirement may be detrimental to other requirements. It also provides the engineers a chance to trace the rationale to requirements and ask customers why the requirements have the values that they do. The Concept Requirement List [Grady, 2006] gives engineers a simple template to follow and helps them focus on the content of the requirement rather than the grammar or sentence structure, which can be fixed after the content is right. Rapid prototyping is used instead of the other methods and includes interviews with the stakeholders. Interviews are generally not sufficient in gathering all of the requirements due to the questioner leaving out important questions. Model Based Systems Engineering (MBSE) [Turk, 2005; Karban, Andolfato, and Zamparelli, 2009] allows engineers to model the system and have the parts of the system trace back to the requirements, but the requirements are still textual statements that are linked to model elements. Planguage [Gilb, 2010] can be used to collect the background information that is associated with a requirement so engineers have the context for the requirements and can understand them better. Planguage can be used with Model Based Systems Engineering. The two together would give engineers the background information they need to better understand not just the requirements but the purpose and value of the requirements. The Object-Process Methodology (OPM) [Dori, 2002] allows system behavior to be captured in a single diagram type that keeps text consistent with graphics. Requirements can be discovered with this methodology and linked to objects and processes. Agile focuses on fast software development so the customer receives the product in frequent, small deliveries that are highly adaptable. Lean Systems Engineering improves processes and eliminates waste. Design for Six Sigma has transformation techniques and metrics for tracking how well individual requirements are met.

While there are lists for the characteristics of good requirements, there is no method for evaluating requirements against those characteristics. There is also no methodology or technique to rate a requirement or track how close the requirement is to becoming a quality requirement as it is developed over time.

4. CONCLUSION

A few of the sources for requirements are from software development programs and software engineering journals. Even books and journals titled with a more generic name of requirements engineering focus on IT projects. There does not seem to be much guidance for top-level system requirements development. While there are many lists of defining what is a good requirement and many ideas on facilitating discussions with the customer and ways to trace requirement changes for impact analysis, the actual formulation of a requirement seems fuzzy. First, engineers and analysts are taught to hold discussions with the customer to find out what is important. Then, the next step in the process is to write down the requirement and engineers are taught to use a style and told not to use certain vague words. Point in fact, there are a few techniques for prioritizing requirements. Likewise, there are many techniques for facilitation. Furthermore, once the requirement is written, there are ways to trace that requirement to the design elements. In contrast, one of the major problems with requirements is ambiguity, but many seem to think that ambiguity is simply about vague words such as appropriate and when necessary, but it is more than that. Ambiguity means there is more than one way to interpret a requirement and calls for more than a dirty word check to resolve. Lastly, there is no literature on quantifying how good a requirement specification is, based on the quality of the requirements taken as a whole.

While it is necessary for requirements to evolve over time, as issues are understood better and threats change, engineers and customers need to apply due diligence in defining the requirements, making sure everyone understands them and agrees with them, and checking that they are feasible. When a program needs to make changes to the requirements, those changes also need to follow good requirements development processes in addition to change management processes.

In summary, many of the techniques fall short in describing the actual process of transitioning from the requirements identification stage to well-written requirements that are complete and unambiguous. What is needed is a requirements engineering framework for complex systems. The framework needs to score the quality of a requirement and the set of requirements. Finally, we need a way to define the nature of the interdependencies among requirements, external systems, and the environment to assist with optimization, sensitivity analyses, and determining the impacts of changing a requirement or system characteristic. In conclusion, the current state of requirements engineering for complex systems includes many ways to elicit and document requirements, but lacks guidance on transforming customer needs into precise, complete, unambiguous requirements.

REFERENCES

A. AlAli and A. Issa, Towards well documented and designed agile software development, World Academy of Science, Engineering and Technology, Las Cruces, NM, March 2011, Vol. 73.

L. Balmelli, B. Nolan, B. Brown, T. Bohn, and U. Wahli, Model driven systems development with rational products, IBM Redbooks, Armonk, NY, 2008.

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, B. Jeffries, J. Kern, B. Marick, R.C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, Manifesto for agile software development, http://agilemanifesto.org/iso/en/, accessed January 2012.

K. Bittner, When requirements go bad, An Ivar Jacobson Consulting White Paper, Alexandria, VA, January 2008.

A. Blekhman, D. Dori, and R. Martin, Model-based standards authoring, Proc 21st INCOSE Int Symp, Denver, CO, June 19–23, 2011.

S. Childers and J. Long, A concurrent methodology for the systems engineering design process, Vi Tech, Blacksburg, VA, 1994.

J. Colombi and R. Cobb, Application of systems engineering to rapid prototyping for close air support, Defense Acquisition University, Fort Belvoir, VA, October 2009.

J. Daniels, T. Bahill, and R. Botta, A hybrid requirements capture process, Symp Proc Fifteenth Annu Int Symp INCOSE, 2005.

DAU, Systems engineering fundamentals guide, Defense Acquisition University Press, Fort Belvoir, VA, 2001.

A. Davis, Software requirements: Objects, functions, and states, second edition, Prentice Hall, Upper Saddle River, NJ, 1993.

D. Dori, Object-process methodology—a holistic systems paradigm, Springer, New York, 2002.

C. Errey, Getting requirements right the first time (Part 1), The Performance Technologies Group, Sydney, Australia, March 2007, http://www.ptg-global.com/PDFArticles/Getting requirements right the first time (Part 1) v1.0.pdf.

G.H. Fisher, Model-based systems engineering of automotive systems, Digital Avionics Syst Conf 1998, Proc 17th DASC, October–November 1 (1998), B15/1–B15/7.

T. Gilb, Competitive Engineering: A handbook for systems engineering, requirements engineering, and software engineering using planguage. Elsevier Butterworth–Heinemann, Amsterdam, 2005, Chapter 5.

T. Gilb. What's wrong with requirements specification? An analysis of the fundamental failings of conventional thinking about software requirements, and some suggestions for getting it right, J Software Eng Appl 3 (2010), 827–838.

J.O. Grady, System requirements analysis, Elsevier, Amsterdam, 2006.

Y. Grobshtein, V. Perelman, E. Safra, and D. Dori, Systems modeling languages: OPM versus SysML, Proc 2007 Int Conf Syst Eng Model, 2007, pp. 102–109.

H. Hoffmann, Systems engineering best practices with the rational workbench for systems and software engineering harmony deskbook release 3.1.1, IBM, Armonk, NY, 2010.

C. Hood, S. Wiedemann, S. Fichtinger, and U. Pautz, Requirements management, the interfaces between requirements development and all other systems engineering processes. Springer-Verlag, Heidelberg, 2008.

I. Hooks, Writing good requirements, Proc Third Int Symp NCOSE, 1993, Vol. 2.

E. Hull, K. Jackson, and J. Dick, Requirements engineering, third edition, Springer-Verlag, Heidelberg, 2011.

IEEE, IEEE Std 610.12-1990, Standard glossary of software engineering terminology, New York, 1990.

IEEE, IEEE 830-1998, IEEE recommended practice for software requirements specifications, New York, 1998.

INCOSE, Requirements engineering guide for all, Seattle, WA, 2011a, http://www.incose.org/REGAL/Regal.aspx, accessed November 2011.

INCOSE, Systems engineering handbook, Seattle, WA, 2011b.

R. Karban, L. Andolfato, and M. Zamparelli. Towards model re-usability for the development of telescope control systems, Int Conf Accelerator Large Exp Phys Control Syst, October 2009.

J. Kasser, The First Requirements Elucidator Demonstration (FRED) tool, Syst Eng 7(3) (2004), 243–256.

J. Kasser, M. Weiss, and A. Hari, Lessons learnt from the applications of QFD to the definition of complex systems, Proc 16th Annu Int Symp Int Council Syst Eng, 2006.

H. Khurana and J.S. Sohal, Agile: The necessitate of contemporary software developers, Int J Eng Sci Technol 3(2) (2011), 1031–1039.

P. Laplante, Requirements engineering for software and systems, Auerbach, New York, 2009.

Lean Enterprise Institute Forum (LEIF), Cambridge, MA, http://www.lean.org/fusetalk/forum/ messageview.cfm?catid=44\&threadid=3866, accessed June 2011.

NDIA Task Group, Top five systems engineering issues in defense industry, National Defense Industrial Association, Systems Engineering Division, Arlington, VA, January 2003.

NDIA Task Group, Top five systems engineering issues within Department of Defense and defense industry, National Defense Industrial Association, Systems Engineering Division, Arlington, VA, July 2006.

NDIA Task Group, Top systems engineering issues in US defense industry, National Defense Industrial Association, Systems Engineering Division, Arlington, VA, September 2010.

B. Oppenheim, Lean for systems engineering with lean enablers for systems engineering, Wiley, Hoboken, NJ, 2011.

B. Oppenheim, E. Murman, and D. Secor, Lean enablers for systems engineering, Syst Eng 14(1) (2010), 29–55.

K. Pohl, Requirements engineering: Fundamentals, principles, and techniques, Springer-Verlag, Heidelberg, 2010.

Software Engineering Institute (SEI), Architecture tradeoff analysis method, Pittsburgh, PA, http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm, accessed February 2012.

F.T. Sheldon, K. Kavi, J. Yu, W. Everett, R. Tausworthe, and R. Brettschneider, Reliability measurement: From theory to practice, IEEE Software 9(4) (July 1992), 13–20.

N. Sledge, Defense acquisition in an unaccountable world, Natl Defense (May 2011).

The Standish Group International, The chaos report, Boston, MA, 1995.

A. Stone, S. Rahmani, W. Luk, and S. Sweet. Rapid prototyping via automatic software code generation from formal specifications: A case study, Symp Proc Second Annu Int Symp Natl Council Syst Eng, 1992.

B. Thomas, Meeting the challenges of requirements engineering, SEI Interactive (March 1999).

D.M.M. Thompson and M.H. Fallah, QFD—a starting point for customer satisfaction metrics, World Prosperity Commun, IEEE Int Conf,1989, Vol.3, pp. 1324–1328.

W. Turk, Requirements management: A template for success, Defense AT&L (March–April 2005).

A. van Lamsweerde, Requirements engineering: From system goals to UML models to software specifications, Wiley, Hoboken, NJ, 2009.

J. Wagner, An implementation of the Analytic Hierarchy Process (AHP) on a large scale integrated launch vehicle avionics systems engineering architecture trade study, Proc Ninth Annu Int Symp Int Council Syst Eng, 1999.

K.E. Wiegers, Writing quality requirements, Software Developer (May 1999). http://www.cs.bgu.ac.il/~elhadad/se/requirements-wiegers-sd

K. Yang and B. El-Haik, Design for Six Sigma: A roadmap for product development, McGraw-Hill, New York, 2009.

Yvonne Bijan has worked for Lockheed Martin for over 12 years and is working on her Ph.D. in System Engineering at Southern Methodist University. She has worked on numerous aeronautic programs including F-35 and C-130J. Ms. Bijan has also worked on the Space Based Infrared System. She is a Certified Enterprise Architect, Certified Systems Engineering Professional, Certified SysML Model Builder Advanced, and holds a QFD Greenbelt. She has a B.S. in Physics with a minor in Mathematics, Colorado State University, Fort Collins, CO and an M.S. in Computer Science, Colorado Technical University, Greenwood Village, CO with a concentration in Software Engineering.

Junfang Yu is an Assistant Professor in the Department of Engineering Management, Information, and Systems at Southern Methodist University. Before joining SMU, he worked as a Supply Chain and Logistics solution architect at i2 Technologies, Dallas, TX and as an Assistant Professor at Louisiana State University, Baton Rouge, LA. He had also worked as a faculty member at Shanghai Jiao Tong University, China. His research interests are in the areas of Supply Chain and Logistics Systems Engineering and Optimization. He has done many research and consulting projects with companies like Sandia National Laboratory, IBM, Northrop Grumman, Honeywell, etc.

Jerrell Stracener is Associate Professor and founding Director of the Southern Methodist University Systems Engineering Program. He teaches courses in systems analysis methods and applications, and directs and conducts systems engineering research. He is the SMU Lead Senior Researcher in the DoD-sponsored Systems Engineering Research Center. Previously he conducted and directed systems engineering studies and analysis on many of the nation's most advanced military aircraft at LTV/Vought/Northrop Grumman. Jerrell was cofounder and leader of the SAE Reliability, Maintainability and Supportability Division. He is an SAE Fellow. Dr Stracener earned Ph.D. and M.S. degrees in Statistics from SMU.

Timothy Woods received his M.S. in Engineering Management, M.S. in Systems Engineering, and Ph.D. in Applied Science with a concentration in Systems Engineering from Southern Methodist University and his bachelor's degree in Electrical Engineering from Michigan Technological University, Houghton, MI. He has worked in the aircraft industries for 21 years and as a Systems Engineering Consultant for 3 years. He currently works at Lockheed Martin Aeronautics, Fort Worth, TX in the Air System Design and Integration Engineering Core leading the Air System Analysis Team.